定向模糊测试技术综述

张月明[1)] 张云菲[1)] 沈小茜[1)] 孙宇帆[1)]

[1)] (南京大学软件学院 南京 210093)

摘 要 模糊测试技术是一种应用模糊过程来验证待测程序（PUT）是否违反正确性策略的测试技术，被广泛应用于自动化漏洞挖掘。然而，模糊测试技术在生成测试用例的过程中，由于搜索空间过大，因此存在着较大的随机性和盲目性，无法有效率地找到潜在的漏洞。定向模糊测试技术通过定位到目标位置，提高了模糊测试的效率，被广泛应用于缺陷复现、补丁检验和静态分析验证。本文通过调研大量文献，分析了定向模糊测试的特征以及常见的模型，总结了多篇论文中定向模糊测试在目标识别、适应度、模糊优化方面使用的度量指标，阐述了当今定向模糊测试的几种应用场景，并在最后探究了若干未来可能的发展方向。

关键词 定向模糊测试；种子选择；调度方式；缺陷复现

A Survey on Directed Fuzzing Technology

ZHANG Yue-Ming[1)] ZHANG Yun-Fei[1)] SHEN Xiao-Qian[1)] SUN Yu-Fan[1)]

[1)](Software Institute, Nanjing University, Nanjing 210093)

Abstract Fuzz testing technology is a kind of testing technology that uses fuzzing process to verify whether the program under test (PUT) violates the correctness strategy. It is widely used in automatic vulnerability mining. However, in the process of generating test cases, the fuzzing technology has a large randomness and blindness due to the large search space, so it cannot find the potential loopholes efficiently. Directed fuzzing technology improves the efficiency of fuzzing testing by locating the target position, and is widely used in crash reproduction, patch testing and static analysis verification. Based on the research of a large number of literatures, this paper analyzes the characteristics and common models of directed fuzzing, summarizes the metrics used by directed fuzzing in the aspects of target identification, fitness and fuzzing optimization in many papers, describes several current application of directed fuzzing, and finally explores some possible development directions in the future.

Keywords Directed fuzzing; seed selection; scheduling strategy; crash reproduction

1 引言

1.1 应用背景

安全测试是现代软件最有效的漏洞检测技术之一。在安全测试技术中，模糊测试被认为是最有效且可扩展的，它为 PUT 提供各种输入，并监控异常行为，例如堆栈或缓冲区溢出、无效读/写、断言故障或内存泄漏。自提出以来，模糊测试越来越受行业和学术界欢迎，并演变为适用于不同测试场景的不同类型的模糊器。

1.2 模糊测试的方法，特点，不足之处

根据模糊测试对 PUT 内部结构的认识，模糊器可以分为黑盒、白盒或灰盒。以灰盒为例，灰盒模糊器基于来自 PUT 执行的反馈信息，使用进化算法来生成新的输入和探索路径。一般来说，模糊器的目标是在有限的时间内覆盖尽可能多的程序状

态。这是因为从直觉上来说，代码覆盖率与 bug 覆盖率密切相关，代码覆盖率高的模糊器可以发现更多的 bug。然而，存在几种测试场景，其中只涉及部分需要进行充分测试的程序状态。例如，如果 MJS（嵌入式设备的 JavaScript 引擎）在 MSP432 ARM 平台上发现了漏洞，则在其他平台的相应代码中可能会出现类似的漏洞。在这种情况下，应该引导模糊器在这些重要的位置重现错误。另一种情况是，当 bug 被修复时，程序员需要检查补丁是否完全修复了错误。这需要模糊器聚焦于那些补丁代码上。在这两种情况下，都需要引导模糊器到达 PUT 中的某些指定位置。

1.3 定向模糊测试的发展

目前，定向模糊测试已经成为一个研究热点，发展非常迅速。它已经超越了依赖手动标记的目标站点，基于距离的度量来区分种子优先级的原始模式。使用了新的适合度指标，例如踪迹相似性和漏洞预测模型。当前的定向模糊测试工具不仅能自动识别目标，而且能以定向的方式暴露目标程序的行为。在不同的场景下，大量的变体已经被应用于软件测试，例如补丁测试[13][20]，回归测试[3]，错误重现[5]，知识整合，结果验证[7][15]，节能[8]和特殊错误检测[5][8]。虽然定向模糊测试发展迅速且有效，但它也有普遍的局限性和挑战，值得进一步研究。在这种背景下，我们开展综述，总结定向模糊测试研究进展的经验证据。在分析定向模糊测试研究的有点和局限性的基础上，我们尝试找出当前研究中的差别，同时揭示新的研究机会，并提出进一步研究的领域。

2 研究框架

2.1 框架图

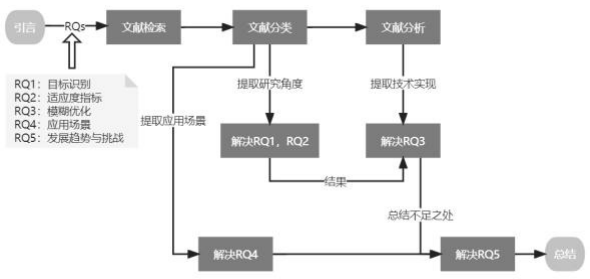

图 1 研究框架图

2.2 研究方法

2.2.1 关键词

根据综述研究的主题，我们选取了如下关键词进行文献检索：定向模糊测试、定向白盒模糊测试、定向灰盒模糊测试。对于白盒模糊器，我们的检索关键词细分为符号执行、程序分析等；对于灰盒模糊器，我们的检索关键词细分为种子选择、调度方式、缺陷复现等。

2.2.2 搜索过程

确定关键词后，我们小组使用以下三种方式搜索相关期刊论文：（1）在软件工程和安全领域的顶级期刊和会议网站上搜索；（2）通过 google scholar 搜索引擎搜索；（3）在 ACM Digital Library 中搜索。随后对收集到的论文进行筛选，下列类型的论文将被排除在外：（1）非 CCF 中 A 类，B 类的会议与期刊；（2）传统的基于覆盖率的模糊测试，非定向模糊测试。（3）论文较短（少于 6 页），只讲模糊器模型，缺少相关工作的背景介绍。（4）较过时的技术选型，如部分 2017 年后的白盒模糊器。

2.2.3 分析过程

将筛选后的论文进行整理，共收集到 27 篇相关论文。其中灰盒部分 21 篇，白盒部分 6 篇。2020-2022 年 12 篇，2017-2019 年 10 篇，2009 年-2016 年 5 篇。根据从每篇论文中提取出的摘要、关键词，我们首先将研究方向进行分类。之后对每篇论文的技术模型展开分析，总结其应用场景，阐述其发展趋势与挑战。具体而言，我们对研究问题如下：

RQ1：定向模糊测试有哪些目标识别方法？

RQ2：定向模糊测试有哪些适应度指标？

RQ3：本文研究的定向模糊器对模糊过程具体做了怎样的优化？

RQ4：定向模糊测试有哪些应用场景？

RQ5：定向模糊测试的发展趋势与挑战是怎样的？

3 目标识别

定向模糊测试的目标识别可分为针对目标位置和针对目标 bug 两类。

3.1 针对目标位置

大多数定向模糊策略的难点在于需要对 PUT 进行目标预标记[9][14][19][21]。手动标记依赖于关于目标位置的先验知识，例如在源代码中的行号，或二进制级别的虚拟内存地址，来标记目标并引导执行到所需位置。然而获得这样的先验知识是具有挑战性的，特别是对于二进制代码。为了合理有效地设置目标位置，研究人员会使用辅助元数据，例如 git 提交日志中的代码变更[20]，从错误跟踪中提取的信息[5]，CVE 漏洞描述中的语义[8]，或深度学习模型[7][15]，来识别代码中易受攻击的功能[7][15]，关键位置，语法标记，完整性检查，以及补丁相关分支[3][13]，并将这些易受攻击的代码部分或位置设定为目标。然而，这种目标识别方案仍然依赖于额外的处理信息并在 PUT 上标记目标的工作。当第一次对 PUT 进行模糊测试或缺乏结构良好的信息时，这种方法是不合适的。

为了提高自动化程度，静态分析工具[10][14][17][23][25]被应用于自动发现 PUT 中的潜在危险区域。然而，这些工具通常只适用于特定错误类型和编程语言。另一种定向利用编译器 sanitiser 的遍（pass）来注释 PUT 中的潜在错误，或者进行二进制级比较来识别补丁相关的目标分支[13]。深度学习方法已被用于预测二进制[15]和抽象语法树级别[7]的潜在易受攻击的代码。最后，攻击面识别组件也被用于自动识别定向灰盒模糊测试的易受攻击目标。

3.2 针对目标 bug

定向灰盒模糊测试也可以用作特定错误检测的手段。例如，UAFuzz [5]和 UAFL [6]利用目标操作序列而不是目标站点来寻找释放后使用漏洞，其存储操作（例如，分配、使用和释放存储）必须以特定顺序执行。AFL-HR [11]通过协同进化方法触发了难以显现的缓冲区溢出和整数溢出的错误。Greyhound [8]指示 Wi-Fi 客户端表现出偏离 Wi-Fi 协议的异常行为。对于针对特定错误的定向灰盒模糊测试，不需要在 PUT 中标记目标，这意味着模糊器可以以进化的方式自动识别和触发这样的错误。

4 适应度指标

4.1 基于距离

AFLGo [21]在编译时检测源代码，并根据 PUT 的调用和控制流图中的边数计算到目标基本块的距离。然后在运行时，它聚合每个基本块的距离值，计算平均值来评估种子。它根据距离对种子进行优先排序，并优先选择更接近目标的种子。一些后续的模糊器也更新了此基于距离的方案。例如，TOFU [9]的距离度量被定义为达到目标所需的正确分支决策的数量。对于每个目标，1dVul [13]采用贪婪突变策略，评估从每个测试输入到目标分支的距离，并优先考虑更接近目标的测试输入。SemFuzz [20]将每一个使用输入的运行称为模糊实例，对于每一个实例，通过内置观测器观察执行以测量易受攻击函数与模糊实例的执行轨迹之间的距离。对应于最短距离的输入被选择为新的种子输入，用于另一轮模糊化，直到达到任何易受攻击函数。UAFuzz [5]则使用调用链的距离度量，导致目标函数更有可能同时包含分配函数和释放函数，以检测复杂的行为释放后使用漏洞。不同于在传统的距离计算中使用等权重的基本块，AFLChurn [3]基于基本块最近被改变的时间或改变的频率来分配数字权重，WindRanger [1]则在计算距离时考虑偏差基本块（即，执行轨迹开始偏离目标位置的基本块）。基于距离的方法的一个缺点是，它只关注最短的距离，因此当有多条路径到达同一目标时，较长的选项可能会被忽略，从而导致差异。另一个缺点是在基本块级计算距离时耗费大量的时间，在一些目标程序上，用户报告说仅仅计算距离文件就要花费数小时。

4.2 基于相似度

相似度是由 Chen 等人在 Hawkeye [19]中首次提出的一种度量，它在函数级别上度量种子的执行轨迹与目标的执行轨迹之间的相似度。直觉是，在“预期轨迹”中覆盖更多功能的种子将有更多机会变异并到达目标。Hawkeye [19]将基本块轨迹距离与覆盖函数相似性相结合，用于种子优先级排序和功率调度。LOLLY [16]使用用户指定的程序语句序列作为目标，并将种子覆盖目标序列的能力(即序列覆盖率)作为评估种子的度量。Berry [10]通过考虑目标序列的执行上下文来升级 LOLLY。这增强

了具有“必要节点”的目标序列，并使用目标执行轨迹和增强的目标序列之间的相似性来区分种子的优先级。然后，相似性被扩展到覆盖其他目标形式，例如操作、bug 跟踪和标记位置。形式上，相似性是某个度量的当前状态和目标状态之间的重叠程度，其中度量包括错误跟踪的长度，以及覆盖的位置、覆盖的操作或覆盖的功能的数量。UAFL [6]使用操作序列覆盖来指导测试用例生成，以逐步覆盖可能触发释放后使用漏洞的操作序列。UAFuzz [5]还使用序列感知的目标相似性度量来测量种子的执行和目标自由后使用错误跟踪之间的相似性。相比之下，基于相似性的度量比基于距离的替代方法能够更好地处理多目标拟合。此外，基于相似性的度量可以包括目标之间的关系，例如目标的排序[5]。最后，在基本块级别测量基于距离的度量，这将引入相当大的开销，而基于相似度的度量可以以相对较高的水平提高整体效率。

4.3 基于概率

研究人员通过使用一些基于深度学习的方法，预测函数的易受攻击概率，量化种子到达目标位置的可能性。易受攻击函数中的每个基本块被给予静态易受攻击分数以测量易受攻击概率，对于每个输入的种子，将执行路径上所有基本块的静态易受攻击分数的总和用作适应度分数，优先考虑具有较高分数的输入[15]。FuzzGuard 在模糊器生成大量新输入后利用模型预测每个输入的可达性。为了解决训练的模型可能不适用于新生成的输入的问题，FuzzGuard [12]设计了一种具有代表性的数据选择方法，以从每一轮变异中抽取训练数据，从而最小化采样数据的数量以提高效率。基于概率的方法可以扩展到指定崩溃以外的位置，例如信息的泄露，漏洞利用、特定的漏洞类型和不同的资源使用。

4.4 基于其他

TIFF [18]监视基本块及其执行频率，并根据执行的基本块计算输入的适合度。任何执行新基本块的输入都会被考虑进一步变异。AFLchurn [3]建议同时对所有提交进行模糊处理，但更多（最近）提交中的代码具有更高的优先级。每个基本块都成为目标，都被分配了一个数字权重，以衡量其最近或多久更改一次。CAFL [2]旨在满足一系列约束条件(即目标站点和数据条件的组合)，而不是到达一组目标站点。它将约束的距离定义为给定种子满足约束的程度，并按顺序排列更好地满足约束的种子的优先级。AFL-HR [11]首次使用脆弱性特定的适应度指标来生成和保留接近暴露漏洞的测试输入，采用了一种称为 headroom 的面向漏洞的适应性度量，它指示测试输入在给定漏洞位置暴露难以发现的漏洞（例如，缓冲区或整数溢出）的程度。

5 模糊测试优化

定向模糊测试的几个关键步骤包括种子输入、种子优先级排序、功率调度、变异器调度。下文将依次从这些方面总结模糊优化的策略。

5.1 种子输入优化

在定向模糊测试中，一个好的种子可以引导模糊器更接近目标位置，并且能够提高后期变异过程的性能。Zong 等人的研究表明，平均而言，AFLGo [21]超过 91.7%的输入不能到达目标代码，即无效输入。许多模糊器对此进行了优化以提升效率，其中动态污点分析技术[18][27]和语义信息分析技术[20]可以帮助生成与输入语法有关的有效输入，提高了到达目标代码，尤其是易受攻击的目标或者安全敏感度高的的目标的概率。此外深度学习技术[12]也被广泛应用其中，在执行之前就过滤掉无法到达目标代码的输入，这样可以显著节省实际运行时间。 BEACON [4]使用了轻量级静态分析来编辑修改那些在运行时无法到达目标代码的路径，这样可以在实际的模糊化运行过程中，减少执行超过 80%的路径，节省了运行时间。

5.2 种子优先级排序优化

在定向模糊测试中，为了提高测试效率，往往更接近目标的种子更先接受变异，因此需要给予高优先级。为了实现这一目标，定向模糊测试通过采用不同的适应度指标，基于控制流分析，来实现种子优先级排序。基于距离的方法[5][9][21]是一种传统且常见的方法，该方法首先计算在调用图或者控制流图中基本块到达目标位置的距离，根据距离的远近来区分不同的种子，从而分配能量。基于相似性的方法[16][19]是基于距离的方法的改进，以种子的执行轨迹和目标的执行轨迹间的相似度作为评

判标准，从而分配能量。定向混合模糊化[10][13][22]结合了白盒模糊器的精确性和灰盒模糊器的可扩展性。灰盒模糊器可以对输入变异进行优先排序和调度，以更快速地接近目标，而白盒模糊器利用符号执行策略，可以解决复杂的路径约束，从而获得更精确的路径。

5.3 功率调度优化

在定向模糊测试中，更接近目标的种子不仅需要更高的种子优先级，还需要被分配更多的功率，以产生更多的变异体。大多数定向灰盒模糊器[19][21]使用模拟退火算法来分配功率。传统的随机调度算法总是接受更好的解决方案，这可能陷入局部最优而忽略了全局最优解。而模拟退火算法实现了以一定的概率接受比当前更差的解决方案，因此有一定的概率跳出局部最优解，找到全局最优解。[3]提出了一种基于蚁群优化的字节级功率调度算法，这种算法通过回归的方法，找出最近更改或更频繁更改的代码，因为这些代码中往往有更大的概率存在 bug，因此给这些字节分配更多的功率。[10][16]利用温度阈值优化了基于模拟退火算法，分为探索和开发两个阶段。在探索阶段，随机变异提供的种子以产生许多新的输入，在开发阶段，从具有更高序列覆盖的种子产生更多新的输入。

5.4 编译器调度优化

在定向模糊测试中，可以将变异分为不同的粒度，并根据实际的模糊测试的过程来动态调整变异的粒度，这样可以提高种子变异的精度和速度，从而增强输入的定向性。通常的方法是先将变异子分为粗粒度和细粒度，然后进行动态调整[15][17][19][20]。粗粒度的变异器往往用于批量改变字节，从而将执行目标指向脆弱的代码块。细粒度的变异器往往仅涉及少量字节级的修改、插入或删除，以便监控关键变量。变异器调度是由经验值控制的，随着种子越来越接近目标，模糊器给予粗粒度变异的机会减少，而给予细粒度变异的机会增加。

6 应用场景

6.1 名词解释

6.1.1 缺陷复现

通过将已更改过的语句设置为目标来进行补丁测试。当一个关键组件发生更改时，我们希望检查这是否引入了任何漏洞。一个关注于这些变化的模糊器有更高的机会暴露回归。

6.1.2 补丁检验

通过在堆栈跟踪中设置方法调用作为目标来崩溃复制。当报告现场崩溃时，只有堆栈跟踪和一些环境参数被发送到内部开发团队。为了保护用户的隐私，特定的崩溃输入通常不可用的，定向模糊器使得内部团队可以快速重现这类崩溃。

6.1.3 静态分析报告验证

静态分析报告通过将静态分析工具设置为报告为潜在危险的目标来验证。在实验中，一个工具可能会将程序的某几行定位为潜在的缓冲区溢出，定向模糊器可以生成测试输入，显示漏洞确实存在。[21]

6.2 具体分析

和传统的定向白盒模糊技术和由覆盖引导的灰盒模糊技术 CGF 相比，DGF 把大部分的时间花费在接近目标站点和对目标站点进行测试，减少了一些无关程序组件对测试资源的浪费。这种特性使得 DGF 在缺陷复现、补丁测试和静态分析报告验证这三个应用场景中具有出色的表现：

（1）对于缺陷复现，将崩溃站点设置为目标站点，使用 DGF 技术可以快速复制和获取概念验证性的导致崩溃的输入。

（2）对于补丁测试，将打补丁的代码站点设置为目标站点，使用 DGF 技术可以快速测试补丁是否完整。

（3）对于静态分析报告验证，将静态分析报告的可疑位置设置为目标站点，使用 DGF 技术可以检查报告的位置是否确实存在漏洞。

而后续研究中提出的技术又在 DGF 的基础之上提高了模糊器在处理上述相关场景下的性，下面将对不同技术在不同应用场景中的表现进行比较，用于讨论的技术包括基于符号执行引擎 Klee 的定向白盒模糊器 Katch、BugRedux，定向灰盒模糊技

术 AFLGo、WindRanger，由覆盖引导的灰盒模糊技术 CGF（AFL 和 FairFuzz），根据静态信息和执行跟踪对执行的种子进行评估的 Hawkeye，利用序列定向策略和并行执行技术来提高模糊的有效性的序列定向混合模糊技术（SDHF），结合了基于距离的定向模糊机制和基于主导的定向符号执行机制的补丁检验问题的解决方案 1dVul，序列覆盖定向模糊技术（SCDF）。

6.3 不同技术在缺陷复现中的比较

6.3.1 定向灰盒与非定向灰盒

通过将漏洞位置设置为目标站点，DGF（AFLGo 和 WindRanger）在暴露相对崩溃方面可以获得比 CGFs（AFL 和 FairFuzz）更好的性能。在实验程序的 11 个漏洞中，WindRanger 在第 9 个漏洞上获得了更好的μTTE。总的来说，WindRanger 在 TTE 上对 AFL 和 FairFuzz 中分别提高了 47%和 59%。[1]

6.3.2 定向灰盒与定向白盒

AFLGo 在基准测试上比 BugRedux 有效得多，AFLGo 无法复制的唯一崩溃是由于一个工程问题：AFLGo 无法生成多个文件。[21]

6.3.3 定向灰盒技术之间

与 AFLGo 相比，30%的加速表明 WindRanger 具有更好的定向能力。对于难以暴露的 CVEs（μTTE>1 小时），WindRanger 明显比其他工具快 1.22×到 2.2×。对于 CVE-2016-4491 和 CVE-2016-6131，WindRanger 获得了最多的打击回合对于 CVE-2016-4489 和 CVE-2016-4490，WindRanger 的性能并不如 AFLGo。然而，这些 cve 只需要几分钟就可以很容易暴露。[1]

6.3.4 定向灰盒技术和 Hawkeye

在大多数情况下，Hawkeye 在到达目标地点的时间和暴露崩溃的时间方面都优于最先进的灰盒模糊器。特别是 Hawkeye 曝光某些崩溃的速度比最先进的 AFLGo 快 7 倍，将曝光时间从 3.5 小时减少到 0.5 小时。[19]

6.3.5 序列定向混合模糊技术与定向模糊技术（白盒 BugRedux，灰盒 AFLGo）

对于 BugRedux 的基准测试，Berry 和 AFLGo 都可以重现崩溃，进一步比较这两种工具触发崩溃的时间成本，因为有更轻量的静态分析，Berry 的仪器时间比 AFLGo 减少很多，同时 Berry 的运行时间比 AFLGo 略短，这显示了 Berry 的并行执行的优势[10]。

6.3.6 序列覆盖定向模糊 LOLLY 与 AFLGo

LOLLY 和 AFLGo 在运行时阶段的性能在统计学上具有可比性。然而，在包含检测阶段和运行时阶段的整个运行中，LOLLY 在整体性能方面是优越的。[16]

6.4 不同技术在补丁检验中的比较

6.4.1 定向灰盒与定向白盒

和 Katch 比较，AFLGo 对于之前未被发现的改变的基本区块的覆盖比 Katch 多 13%。AFLGo 可以覆盖 Katch 无法覆盖的 84 个目标，而 Katch 则覆盖了 AFLGo 无法覆盖的 59 个目标。小交集的产生可以归因于每种技术的个别优势，符号执行可以解决进入“隔间”（compartments）的困难约束，否则将很难访问。另一方面，灰盒模糊器可以快速探索许多指向目标的路径，而不会被困在一个特定的路径“邻域”中。[21]

6.4.2 1dVul 与定向灰盒 AFLGo

1dVul 已经成功地从应用程序中确定的 209 个补丁目标中生成了 130 个目标的输入，而 AFLGo 只能在相同的有限时间预算内分别达到 99 个目标。此外，1dVul 的运行速度分别比 AFLGo 快 2.2 倍，并确认了未打补丁程序的 96 个漏洞。[13]

7 发展趋势与挑战

(1) 现今大部分定向模糊器评估性能的指标只有暴露时间，即暴露给定错误的第一个测试用例之前的模糊过程的时间，而往往忽略了额外性能开销

的测量。因此如果能够将符号执行、静态分析、动态污点分析、机器学习等分析技术带来的性能开销考虑在内，可以更全面公平地度量定向模糊器的性能。

(2) 现今大部分定向模糊器在种子优先级排序时依旧基于等权重度量，即控制流图中每个分支的跳转都是等概率的。目前常见的优化方案如下。计算每条路径能够转换为目标路径的概率，据此进行种子的优先级排序。具体分为两步：首先收集路径中所有分支的概率计算路径概率，然后基于蒙特卡罗统计方法估计分支跳转概率。这样做更易到达定向目标位置，使得定向模糊器的性能更高。

(3) 现今大部分定向模糊器都是在源码层面上的测试，例如根据控制流图计算距离等等都非常依赖源代码。二进制级定向模糊器研究较少，因为存在着较大的困难。一是运行开销大，基于仿真器的工具如 QEMU 等通常效率较低。二是目标识别困难， 只能从错误跟踪中提取目标信息，而且二进制代码非常难以阅读。 目前的解决方案如下。一是使用硬件辅助如英特尔 PT 处理器跟踪，减少运行开销。二是基于机器学习的方法，自动识别目标二进制代码。这推动了二进制级定向模糊器的发展。

(4) 现今大部分定向模糊器在多目标任务的测试中，并没有特别关注目标之间的关系。而如果能利用目标之间的关系进行优化，则能够提高测试效率。目前的解决方案如下。基于空间关系：考虑它们是否共享同一个分支，以及它们的相对优先级关系。基于状态空间的位置：考虑两个目标是否共享同一个状态，以及状态转换图上两个状态是否可以相互转换。基于多线程的情况：考虑是否存在不同的线程，它们之间相互影响，导致虽然种子到达了目标位置，但并没有触发 bug。

参 考 文 献

[1] Zhengjie Du, Yuekang Li, Yang Liu, and Bing Mao. Windranger: A directed greybox fuzzer driven by deviation basic blocks. In Proceedings of the 44th International Conference on Software Engineering, ICSE '22, page 2440–2451, New York, NY, USA, 2022. Association for Computing Machinery.

[2] Gwangmu Lee, Woochul Shim, and Byoungyoung Lee. Constraint-guided directed greybox fuzzing. In 30th USENIX Security Symposium (USENIX Security 21), pages 3559–3576. USENIX Association, August 2021.

[3] Xiaogang Zhu and Marcel Böhme. Regression greybox fuzzing. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, CCS '21, page 2169–2182, New York, NY, USA, 2021. Association for Computing Machinery.

[4] Heqing Huang, Yiyuan Guo, Qingkai Shi, Peisen Yao, Rongxin wu, and Charles Zhang. Beacon: Directed grey-box fuzzing with provable path pruning. 05 2022.

[5] Manh-Dung Nguyen, Sébastien Bardin, Richard Bonichon, Roland Groz, and Matthieu Lemerre. Binary-level directed fuzzing for Use-After-Free vulnerabilities. In 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID2020), pages 47–62, San Sebastian, October 2020. USENIX Association.

[6] Haijun Wang, Xiaofei Xie, Yi Li, Cheng Wen, Yuekang Li, Yang Liu, Shengchao Qin, Hongxu Chen, and Yulei Sui. Typestate-guided fuzzer for discovering use-after-free vulnerabilities. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, ICSE '20, page 999–1010, New York, NY, USA, 2020. Association for Computing Machinery.

[7] Xiaogang Zhu, Shigang Liu, Xian Li, Sheng Wen, Jun Zhang, Seyit Ahmet Çamtepe, and Yang Xiang. Defuzz: Deep learning guided directed fuzzing. ArXiv, abs/2010.12149, 2020.

[8] Matheus E. Garbelini, Chundong Wang, and Sudipta Chattopadhyay. Greyhound: Directed greybox wi-fi fuzzing. IEEE Transactions on Dependable and Secure Computing, 19:817–834, 2022.

[9] Zi Wang, Ben Liblit, and T. Reps. Tofu: Target-oriented fuzzer. 2020.

[10] Hongliang Liang, Lin Jiang, Lu Ai, and Jinyi Wei. Sequence directed hybrid fuzzing. ieee international conference on software analysis evolution and reengineering, 2020.

[11] Raveendra Kumar Medicherla, Raghavan Komondoor, and Abhik Roychoudhury. Fitness guided vulnerability detection with greybox fuzzing. In Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, ICSEW'20, page 513–520, New York, NY, USA, 2020. Association for Computing

Machinery.

[12] Peiyuan Zong, Tao Lv, Dawei Wang, Zizhuang Deng, Ruigang Liang, and Kai Chen. FuzzGuard: Filtering out unreachable inputs in directed grey-box fuzzing through deep learning. In 29th USENIX Security Symposium (USENIX Security 20), pages 2255–2269. USENIX Association, August 2020.

[13] Jiaqi Peng, Feng Li, Bingchang Liu, Lili Xu, Binghong Liu, Kai Chen, and Wei Huo. 1dvul: Discovering 1-day vulnerabilities through binary patches. 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pages 605–616, 2019.

[14] Valentin Wüstholz and Maria Christakis. Targeted greybox fuzzing with static lookahead analysis. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, ICSE '20, page 789–800, New York, NY, USA, 2020. Association for Computing Machinery.

[15] Yuwei Li, Shouling Ji, Chenyang Lv, Yuan Chen, Jianhai Chen, Qinchen Gu, and Chunming Wu. V-fuzz: Vulnerability-oriented evolutionary fuzzing. 01 2019.

[16] Hongliang Liang, Yini Zhang, Yue Yu, Zhuosi Xie, and Lin Jiang. Sequence coverage directed greybox fuzzing. In Proceedings of the 27th International Conference on Program Comprehension, ICPC '19, page 249–259. IEEE Press, 2019.

[17] Lingyun Situ, Linzhang Wang, Xuandong Li, Le Guan, Wenhui Zhang, and Peng Liu. Energy distribution matters in greybox fuzzing. In Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings, ICSE '19, page 270–271. IEEE Press, 2019.

[18] Vivek Jain, Sanjay Rawat, Cristiano Giuffrida, and Herbert Bos. Tiff: Using input type inference to improve fuzzing. In Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC '18, page 505–517, New York, NY, USA, 2018. Association for Computing Machinery.

[19] Hongxu Chen, Yinxing Xue, Yuekang Li, Bihuan Chen, Xiaofei Xie, Xiuheng Wu, and Yang Liu. Hawkeye: Towards a desired directed grey-box fuzzer. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18, page 2095–2108, New York, NY, USA, 2018. Association for Computing Machinery.

[20] Wei You, Peiyuan Zong, Kai Chen, XiaoFeng Wang, Xiaojing Liao, Pan Bian, and Bin Liang. Semfuzz: Semantics-based automatic generation of proof-of-concept exploits. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17, page 2139–2154, New York, NY, USA, 2017. Association for Computing Machinery.

[21] Marcel Böhme, Van-Thuan Pham, Manh-Dung Nguyen, and Abhik Roychoudhury. Directed greybox fuzzing. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17, page 2329–2344, New York, NY, USA, 2017. Association for Computing Machinery.

[22] Saahil Ognawala, Thomas Hutzelmann, Eirini Psallida, and Alexander Pretschner. Improving function coverage with munch: A hybrid fuzzing and directed symbolic execution approach. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC '18, page 1475–1482, New York, NY, USA, 2018. Association for Computing Machinery.

[23] Maria Christakis, Peter Müller, and Valentin Wüstholz. Guiding dynamic symbolic execution toward unverified program executions. In Proceedings of the 38th International Conference on Software Engineering, ICSE '16, page 144–155, New York, NY, USA, 2016. Association for Computing Machinery.

[24] Weiguang Wang, Hao Sun, and Qingkai Zeng. Seededfuzz: Selecting and generating seeds for directed fuzzing. In 2016 10th International Symposium on Theoretical Aspects of Software Engineering (TASE), pages 49–56, 2016.

[25] Brian Pak. Hybrid fuzz testing: Discovering software bugs via fuzzing and symbolic execution. 2012.

[26] Tielei Wang, Tao Wei, Guofei Gu, and Wei Zou. Taintscope: A checksum-aware directed fuzzing tool for automatic software vulnerability detection. 2010 IEEE Symposium on Security and Privacy, pages 497–512, 2010.

[27] Vijay Ganesh, Tim Leek, and Martin C. Rinard. Taint-based directed whitebox fuzzing. 2009 IEEE 31st International Conference on Software Engineering, pages 474–484, 2009.